Semester Long Internship Report

On

UI/UX Design and Implementation of Dynamic Frontend Interface for Yaksh Online Assessment Platform

Submitted by

Mohit Rana

3rd Year B.Sc. IT Student, Department of Computer Science

Graphic Era Hill University

Dehradun

Under the Guidance of

Prof. Prabhu Ramachandran

FOSSEE, Indian Institute of Technology Bombay

Mentors:

Dr.Kushal Shah

Mr.Prathamesh Salunke

July 12, 2026

Acknowledgments

I would like to express my sincere gratitude to everyone who supported me throughout this semester long internship and contributed to the successful completion of my project. This has been an exceptional learning experience, and it would not have been possible without the guidance and support of many individuals and organizations.

- I extend my heartfelt thanks to the project staff at the Python project(yaksh) team, especially **Dr.Kushal Shah**, and **Mr.Prathamesh Salunke**, for their invaluable technical guidance, patience, and continuous support throughout the project.
- I am extremely grateful to the Python project(yaksh) Principal Investigator (PI), **Prof. Prabhu Ramachandran**, FOSSEE, IIT Bombay, for his visionary leadership and support.
- I sincerely thank the FOSSEE Principal Investigator, **Prof. Kannan M. Moudgalya**, Department of Chemical Engineering, IIT Bombay, for enabling such a valuable open-source initiative through this fellowship.
- My appreciation also goes to the FOSSEE Managers, **Ms. Usha Viswanathan** and **Ms. Vineeta Parmar**, and their entire team for their smooth coordination and timely assistance throughout the duration of the program.
- I gratefully acknowledge the support from the **National Mission on Education through Information and Communication Technology (ICT), Ministry of Education (MoE), Government of India**, for facilitating and funding this project and promoting open-source software for education.
- I would also like to thank my fellow interns and colleagues who collaborated with me and shared knowledge throughout this journey.

- Lastly, I wish to acknowledge the support from my college, **Graphic Era Hill University**, the Department of Information Technology, my professors, and the administration for encouraging me to participate in this fellowship and supporting my learning endeavors.

Contents

Chapter 1

Introduction

1.1 National Mission in Education through ICT

The National Mission on Education through ICT (NMEICT) is a scheme under the Department of Higher Education, Ministry of Education, Government of India. It aims to leverage the potential of ICT to enhance teaching and learning in Higher Education Institutions in an anytime-anywhere mode.

The mission aligns with the three cardinal principles of the Education Policy—**access, equity, and quality**—by:

- Providing connectivity and affordable access devices for learners and institutions.
- Generating high-quality e-content free of cost.

NMEICT seeks to bridge the digital divide by empowering learners and teachers in urban and rural areas, fostering inclusivity in the knowledge economy. Key focus areas include:

- Development of e-learning pedagogies and virtual laboratories.
- Online testing, certification, and mentorship through accessible platforms like EduSAT and DTH.
- Training and empowering teachers to adopt ICT-based teaching methods.

For further details, visit the official website: www.nmeict.ac.in.

1.1.1 ICT Initiatives of MoE

The Ministry of Education (MoE) has launched several ICT initiatives aimed at students, researchers, and institutions. The table below summarizes the key details:

No.	Resource	For Students/Researchers	For Institutions
Audio-Video e-content			
1	SWAYAM	Earn credit via online courses	Develop and host courses; accept credits
2	SWAYAMPRABHA	Access 24x7 TV programs	Enable SWAYAMPRABHA viewing facilities
Digital Content Access			
3	National Digital Library	Access e-content in multiple disciplines	List e-content; form NDL Clubs
4	e-PG Pathshala	Access free books and e-content	Host e-books
5	Shodhganga	Access Indian research theses	List institutional theses
6	e-ShodhSindhu	Access full-text e-resources	Access e-resources for institutions
Hands-on Learning			
7	e-Yantra	Hands-on embedded systems training	Create e-Yantra labs with IIT Bombay
8	FOSSEE	Volunteer for open-source software	Run labs with open-source software
9	Spoken Tutorial	Learn IT skills via tutorials	Provide self-learning IT content
10	Virtual Labs	Perform online experiments	Develop curriculum-based experiments
E-Governance			
11	SAMARTH ERP	Manage student lifecycle digitally	Enable institutional e-governance
Tracking and Research Tools			
12	VIDWAN	Register and access experts	Monitor faculty research outcomes
13	Shodh Shuddhi	Ensure plagiarism-free work	Improve research quality and reputation
14	Academic Bank of Credits	Store and transfer credits	Facilitate credit redemption

Table 1.1: Summary of ICT Initiatives by the Ministry of Education

1.2 FOSSEE Project

The FOSSEE (Free/Libre and Open Source Software for Education) project promotes the use of FLOSS tools in academia and research. It is part of the National Mission on Education through Information and Communication Technology (NMEICT), Ministry of Education (MoE), Government of India.

1.2.1 Projects and Activities

The FOSSEE Project supports the use of various FLOSS tools to enhance education and research. Key activities include:

- **Textbook Companion**: Porting solved examples from textbooks using FLOSS.
- **Lab Migration**: Facilitating the migration of proprietary labs to FLOSS alternatives.
- **Niche Software Activities**: Specialized activities to promote niche software tools.
- **Forums**: Providing a collaborative space for users.
- **Workshops and Conferences**: Organizing events to train and inform users.

1.2.2 Fellowships

FOSSEE offers various internship and fellowship opportunities for students:

- Winter Internship
- Summer Fellowship
- Semester-Long Internship

Students from any degree and academic stage can apply for these internships. Selection is based on the completion of screening tasks involving programming, scientific computing, or data collection that benefit the FLOSS community. These tasks are designed to be completed within a week.

For more details, visit the official FOSSEE website.

Figure 1.1: FOSSEE Projects and Activities

1.3 The Yaksh Platform

Yaksh is an open-source online test and grading platform developed and maintained by the FOSSEE team at IIT Bombay. It is designed to conduct online assessments, evaluate programming assignments across multiple languages including Python, Java, C++, and Bash, and manage course grading efficiently. Yaksh is widely deployed in Indian educational institutions for conducting programming courses and evaluations at scale.

Historically, Yaksh has been built as a monolithic, server-side rendered application using the Django web framework. While robust and functionally rich, the reliance on Django's templating engine presented growing limitations: every user interaction required a full page reload, complex UI states were difficult to manage with vanilla JavaScript,

and the overall experience felt dated compared to modern web applications.

1.3.1 Yaksh Architecture & Django Interface

The traditional Yaksh platform utilizes Django's built-in templating engine to deliver a monolithic frontend experience. Key aspects of this interface include:

- **Server-Side Rendering (SSR)**: The UI heavily relies on server-rendered HTML pages, meaning navigation and form submissions (such as running code) traditionally prompt a full page reload.
- **Vanilla JavaScript UI Management**: Dynamic aspects like the coding environment and quiz timers are handled via vanilla JavaScript directly embedded within Django templates.
- **Integrated Evaluation**: The frontend is tightly coupled with the backend grading engine, presenting standard inputs, compiler outputs, and test-case results directly upon page refresh.
- **Role-Based Dashboards**: The interface provides distinct, integrated views for Students (taking assessments) and Moderators/Teachers (creating courses and grading).

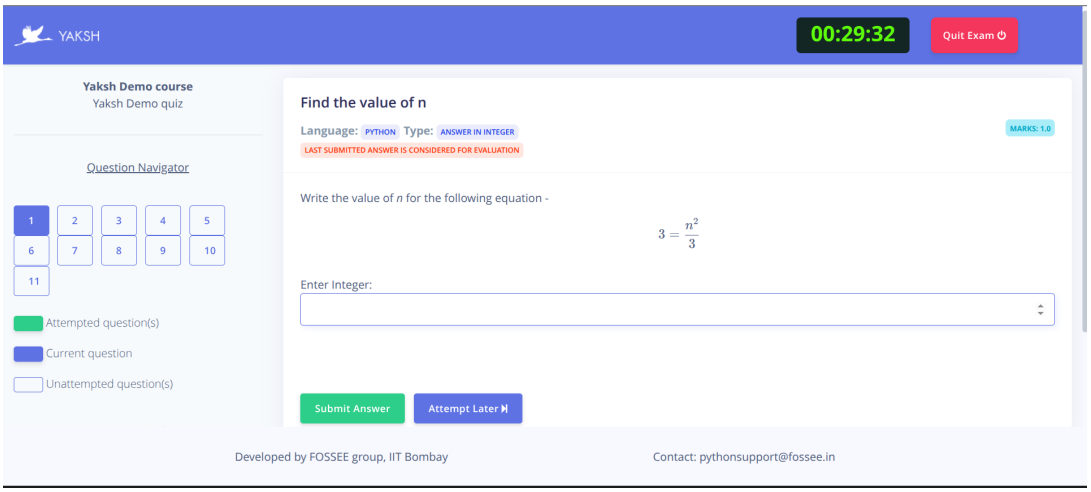


Figure 1.2: Yaksh Platform - Django-based Assessment Interface

1.3.2 Features

- **Multi-Language Evaluation**: Supports automated evaluation of code submissions in languages like Python, Java, C++, and Bash.
- **Scalable Deployments**: Robust architecture capable of handling large-scale, concurrent test-taking environments across institutions.
- **Course & Question Management**: Facilitates efficient management of courses, module-wise enrollments, and diverse question types (MCQs, coding).
- **Automated Grading System**: Generates detailed feedback and standardizes the grading process efficiently without manual intervention.

For more details, visit the official FOSSEE website [1].

Chapter 2

Screening Task

2.1 Problem Statement

Redesign the existing FOSSEE Workshop Booking website to improve performance, modern UI, responsiveness, accessibility, and SEO using React. The goal is to enhance the UI/UX—especially for mobile users—while maintaining fast load times and keeping the core functionality intact.

2.2 Tasks Done

For the Workshop Booking Portal project, I completely modernized a legacy Django-only application into a robust React and Django REST API architecture, significantly improving both user experience and long-term maintainability.

Backend

- Upgraded the entire backend from Django 3.0.7 to 5.2.6, resolving deprecated APIs, breaking changes, and updating all dependencies.
- Transitioned the backend architecture to serve data via a Django REST API instead of traditional server-side rendering.
- Improved overall system security and leveraged performance optimizations introduced in modern Django versions.

Frontend

- Built a completely new, modern user interface using **React** and **Tailwind CSS**.
- Adopted a **mobile-first design philosophy**, ensuring readable navigation, touch-friendly forms, and adaptive layouts across all screen sizes.
- Improved accessibility by enforcing proper color contrast and establishing a clear, consistent visual hierarchy.
- Optimized performance by keeping the JS bundle under 100KB, implementing lazy loading, and prioritizing critical content delivery.

Design & Problem Solving

- Established a unified design system (8px grid, consistent typography) to systematically fix severe layout inconsistencies found in the legacy codebase.
- Balanced rich design with performance by prioritizing efficient CSS transforms over complex animations, ensuring fast sub-2-second load times on mobile networks.

In summary, I successfully transformed the Workshop Booking Portal into a modern full-stack application. By upgrading the legacy backend and implementing a highly responsive, performance-optimized React frontend, I delivered a significantly faster, accessible, and cohesive user experience.

Chapter 3

Task 2: UI/UX Design and Prototyping Phase

3.1 Task 2: Problem Statement

The objective of this phase was to redesign and prototype the user interface of the **Yaksh** platform — a web-based assessment and learning management system. The existing system was built on a legacy Django template-based (server-rendered) architecture, which introduced multiple usability and performance issues for both instructors and students. The task required reimagining the complete user experience from first principles before any application code was written, ensuring the new React SPA-based design would be maintainable, accessible, and significantly more user-friendly than its predecessor.

3.2 Task 2: Tasks Done

1. User Research and Journey Mapping

The initial step involved thoroughly reviewing the existing Django template-based interface to identify all current workflows, pain points, and missing features for the two primary user roles — **Instructors** and **Students**.

a. Instructor Journey:

- Identified the instructor workflow as the most data-intensive use case.

- Mapped requirements for course management dashboards, quiz builders, grading configuration panels, and live student activity monitoring.
- Key pain points uncovered: no drag-and-drop question arrangement, cumbersome multi-step programming question creation flow, and no live view of students currently attempting a quiz.

b. Student Journey:

- Focused on designing a distraction-free, focused assessment environment.
- Identified critical UX issues in the legacy system: absence of a persistent quiz navigation sidebar, forced linear progression through questions, full-page browser reloads disrupting the exam timer, and a weak post-examination feedback mechanism with no side-by-side code comparison.
- Designed solutions: a persistent `QuizSidebar` component for non-linear navigation, async answer validation, and a `ViewAnswerPaper` page for post-exam review.

2. Wireframing

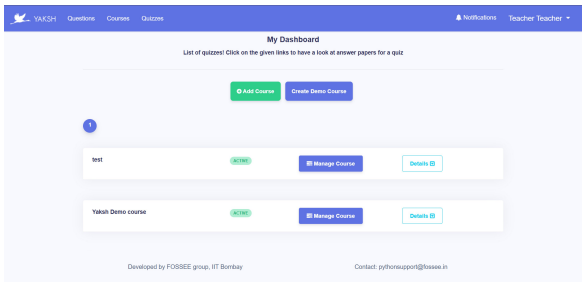
- Produced rough wireframes in **Figma** to establish the basic layout direction — sidebar positioning, navigation structure, and main screen layouts.
- Wireframes were treated as a guiding starting point rather than a fixed specification, allowing layouts to evolve during frontend development.
- **Figma Link:** YAKSH-WIREFRAMES-TEACHER (Figma)

The figures below illustrate the contrast between the legacy Django template-based interface and the redesigned Figma wireframes produced during this phase.

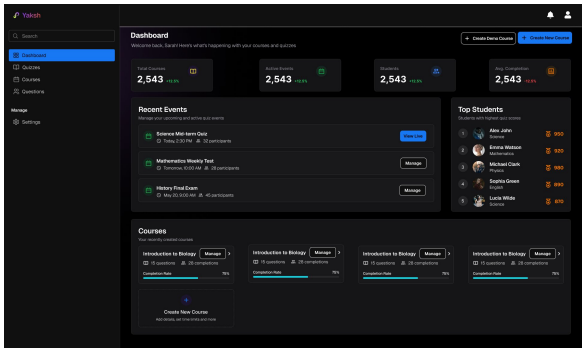
3. Prototyping

A more refined prototype was built covering the following key screens and user flows:

- **Authentication Flow:** Login, Signup, and Forgot Password screens with input validation state indicators.

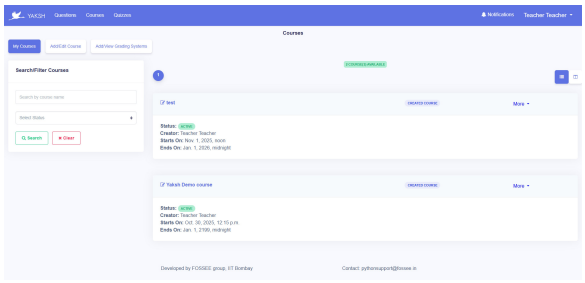


Before: Legacy Django teacher dashboard

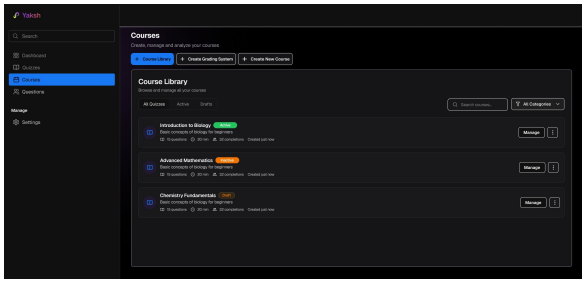


After: Redesigned Figma wireframe — Teacher Dashboard

Figure 3.1: Teacher Dashboard — Legacy UI vs. Figma Wireframe



Before: Legacy Django courses page



After: Redesigned Figma wireframe — Courses Page

Figure 3.2: Courses Page — Legacy UI vs. Figma Wireframe

- **Instructor Interface:** Course management overview, course design canvas, question bank browser, grading system configuration panel, and the Live Quiz Monitor dashboard.
- **Student Interface:** Course module listing, active lesson/quiz/exercise viewer, and the quiz enrollment page.
- **Active Quiz Interface:** Distraction-minimized layout with a code editor panel, question navigation sidebar, countdown timer, and submission confirmation modal.
- **Post-Exam View:** `ViewAnswerPaper` screen showing a side-by-side diff of submitted student code versus the correct solution with assertion results.

4. Design Tokens and Component Library

a. Typography and Color Palette:

- Defined a standardized typographic scale covering heading levels H1 through H4 and body copy, mapped to a consistent type ramp.
- Established a primary and secondary color palette with both light and dark mode variants to support a `ThemeController` feature.
- Contrast ratios verified against **WCAG AA** accessibility standards.

b. Reusable Component Library:

- Designed a comprehensive library of reusable UI components, each with all required interaction states: default, hover, active, disabled, and error.
- Key components: primary and secondary buttons, form input fields, dropdown selectors, modal overlays, toast notification variants, data table rows, and the course module card.
- The Tailwind CSS configuration was set up during development, with styling decisions refined progressively as components were built.

5. Stakeholder Review and Design Approval

- Walked through key user flows with project supervisors and FOSSEE mentors: a teacher creating a course and building a quiz, and a student enrolling and attempting an assessment.
- Incorporated feedback from the review session — primarily layout adjustments — before receiving approval to proceed into development.
- The final UI continued to evolve iteratively based on practical implementation constraints and ongoing mentor feedback.

3.3 Task 2: Summary of Contribution

- Conducted a comprehensive review of the legacy Yaksh interface and mapped user journeys for both Instructor and Student roles.
- Identified and documented critical UX pain points in the existing server-rendered system.

- Designed and iterated on wireframes and a high-fidelity prototype in Figma covering all major screens and flows.
- Defined design tokens (color palette, typography scale) and a reusable component library with full interaction states.
- Verified accessibility compliance against WCAG AA contrast standards.
- Presented the prototype to supervisors, incorporated feedback, and received approval before development commenced.
- Established the design foundation that guided the entire React SPA frontend implementation.

This structured design phase ensured that the subsequent frontend development was grounded in validated user needs and a clearly approved design direction, significantly reducing rework and aligning the team on the product vision from the outset.

Chapter 4

Task 3: System Architecture and Backend API Development

4.1 Problem Statement

With the UI/UX design phase approved, the next objective was to establish the complete technical foundation for the modernized Yaksh platform. This involved two interconnected concerns: **architecting a clean headless (decoupled) system** and **building a comprehensive RESTful API layer** on top of the existing Django backend using Django REST Framework (DRF). The legacy monolithic system — where the Django templating engine rendered HTML on the server for every user interaction — had to be replaced by a clean separation between the frontend SPA and the backend evaluation engine, without disrupting the well-tested core Yaksh logic.

4.2 Tasks Done

1. Proposed Headless Architecture

The solution adopted was a **Headless (Decoupled) Architecture**. Under this model:

- The **Django backend** handles purely business logic: compiling and evaluating code submissions through existing evaluator modules, managing database state, and dispatching asynchronous tasks through Celery workers.

- A new Django application, `api`, was introduced to expose this logic as a clean RESTful API layer.
- The **React frontend** (built and served by Vite) handles all UI rendering, client-side routing, and state management entirely in the browser — communicating with the backend exclusively through JSON API requests via Axios.

2. Technology Stack

Category	Technology	Rationale
Frontend Core	React 19 + Vite	Ultra-fast HMR and optimized production builds
State Management	Zustand	Lightweight, boilerplate-free global state; chosen over Redux for modular store design
Styling	Tailwind CSS + PostCSS	Utility-first, responsive styling with a clean configuration-driven approach
Routing	React Router DOM v7	Client-side routing with protected route HOC wrappers
HTTP Client	Axios	Request/response interceptors for JWT token attachment and error handling
Code Quality	ESLint + Prettier	Enforced consistent code style and static analysis
Backend API	Django REST Framework	Serializers and ViewSets exposing existing Yaksh models as RESTful JSON endpoints
Dev Environment	WSL 2 + Ubuntu	Yaksh supports only Linux/macOS; WSL provides a compatible environment on Windows
Task Queue	Celery 4.4.2 + Redis	Asynchronous code re-evaluation; Redis serves as the Celery message broker
Python Runtime	Python 3.6–3.8 + Miniconda	Required by Django 3.0.3 and Yaksh's evaluation engine

Table 4.1: Technology Stack for the Yaksh Modernization Project

3. Backend API Development (`api` module)

a. Serializer Design (`serializers.py`):

- Complex serializers were developed for all core entities: `User`, `Profile`, `Course`, `Quiz`, `Question`, `AnswerPaper`, `Badge`, `Notification`, and more.
- `SerializerMethodField` and computed fields were used for dynamic attributes such as badge progress and per-user statistics.
- Dedicated serializers were written for nested and related entities to support the grading and analytics interfaces.
- To mitigate the **N+1 query problem** arising from deeply nested serialization, `select_related` and `prefetch_related` were applied strategically to QuerySets — ensuring a bounded number of database queries regardless of nesting depth.

b. A concrete example — the `AnswerPaperGradingSerializer`:

Listing 4.1: Nested Serializer with N+1 Prevention (api/serializers.py)

```
class AnswerPaperGradingSerializer(serializers.ModelSerializer):
    user = SimpleUserSerializer(read_only=True)
    answers = AnswerDetailSerializer(many=True, read_only=True)
    question_paper = QuestionPaperSerializer(read_only=True)

    class Meta:
        model = AnswerPaper
        fields = ['id', 'user', 'question_paper', 'answers',
                  'marks_obtained', 'percent', 'status',
                  'attempt_number', 'start_time', 'end_time']
```

The corresponding QuerySet in `views.py` collapses all foreign key lookups and reverse relations into just two SQL operations, regardless of the number of students or answers:

Listing 4.2: QuerySet Optimization to prevent N+1 (api/views.py)

```
answerpapers = AnswerPaper.objects.select_related(
    "user", "question_paper"
).prefetch_related('answers').filter(
    course_id=course_id,
    question_paper_id=question_paper.id,
    attempt_number=attempt_number
```

```
).order_by("user__first_name")
```

c. API Views and URL Design (`views.py` & `urls.py`):

- **Class-based views (CBV)** for standard REST resource lists and detail endpoints (`QuizList`, `QuestionDetail`, etc.).
- **Function-based views (FBV)** for complex, multi-step flows such as custom enrollment, password reset, and code execution bridging.
- **Role-centric URL structure:** All 250+ routes are organized under `student/`, `teacher/`, and `common/` prefixes for discoverability and clean access control.
- **Backward compatibility:** Legacy URL routes are preserved with explanatory comments to ensure existing client flows continue to work without breakage.

d. Functional API Modules Implemented:

- **Authentication:** Secure login, JWT token provisioning, and session management endpoints.
- **Resource Management (CRUD):** Endpoints for creating, reading, updating, and deleting courses, modules, quiz configurations, and grading systems.
- **Question Bank:** Endpoints for MCQ, MCC (Multiple Correct Choice), and programming question types with their associated test cases.
- **Code Execution:** Endpoints bridging the API layer to Yaksh's existing code evaluators (`bash_code_evaluator.py`, `cpp_code_evaluator.py`), so a student clicking "Run Code" in the React UI triggers secure backend code execution.
- **Analytics and Grading:** Endpoints exposing aggregated student performance data, submission histories, and grade override capabilities.
- **Forum, Badges, and Notifications:** Dedicated endpoints for course discussion threads, achievement tracking, and notification delivery.

e. Test Coverage (`api/tests.py`):

- Augmented Django REST test suite covering core CRUD operations, role-specific flows, and edge cases.

- Tests validate full resource lifecycle transitions, permission enforcement, and error state handling.

4.3 Code Snippet: Sample API Response

The following is a representative response from the teacher-side question retrieval endpoint (`GET /api/teacher/questions/4/`), demonstrating the structured nested JSON output delivered to the React frontend:

Listing 4.3: Sample API Response for Question Retrieval


```
{
  "id": 4,
  "summary": "Check Palindrome",
  "type": "code",
  "language": "python",
  "points": 2.0,
  "snippet": "def is_palindrome(s):",
  "test_cases": [
    {
      "id": 5,
      "type": "standardtestcase",
      "test_case": "assert is_palindrome('hello') == False",
      "weight": 1.0,
      "hidden": false
    },
    {
      "id": 6,
      "type": "standardtestcase",
      "test_case": "assert is_palindrome('nitin') == True",
      "weight": 1.0,
      "hidden": false
    }
  ]
}
```

4.4 Summary of Contribution

- Designed and implemented the headless decoupled architecture separating the React SPA from the Django evaluation engine.
- Built the full `api` Django application housing all serializers, views, and URL configurations.
- Developed complex nested serializers with N+1 query prevention using `select_related` and `prefetch_related`.
- Implemented 250+ RESTful endpoints organized by role (student/teacher/common) with backward-compatible legacy route preservation.
- Wrote comprehensive backend test coverage for CRUD operations, permissions, and resource lifecycle flows.
- Configured environment-variable-driven Django settings for clean separation of development and production configuration.

Chapter 5

Task 4: Frontend SPA Development

5.1 Problem Statement

With the API layer in place, the objective of this phase was to architect and implement the complete **React Single Page Application (SPA)** for the Yaksh platform. The frontend had to serve two functionally distinct user roles — **Instructors** and **Students** — each with complex workflows, while sharing a consistent design language and navigation structure. The implementation also had to address two non-trivial technical challenges: a legacy MCQ schema mismatch between the backend database and the React data model, and the need for an isolated instructor quiz-testing sandbox that would not contaminate live course analytics.

5.2 Tasks Done

1. State Management Architecture (`src/store/`)

Rather than a monolithic global state store, a **modular, domain-scoped Zustand architecture** was implemented to ensure UI re-renders are scoped precisely to the components that consume changed state:

- `authStore.js` & `userStore.js`: Manage persistent user sessions using `localStorage`, role identification (student vs. teacher), and state required for UI access control throughout the application.

- `questionsStore.js` & `quiz_QuestionStore.js`: Handle the lifecycle of caching the question bank, reducing redundant API calls during dynamic quiz assembly by the instructor.
- `quizMonitorStore.js`: Manages near-real-time state of students actively attempting a quiz, feeding live metrics to the instructor's Quiz Monitor Panel.
- `quizRegradeStore.js`: Handles instructor-initiated grade override state, decoupled from the primary quiz flow stores.
- `forumStore.js`: Manages discussion thread state for the course discussion forum feature.

2. Routing and Access Control

- **Role-based protected routes** implemented via `PrivateRoute` and `PublicRoute` HOC wrappers.
- Teacher and Student portal areas are cleanly separated through distinct route layouts and logic.
- Deep-linking and parameterized routes support direct resource access (e.g., `/courses/:courseId/`
- JWT-secured Axios interceptors handle token attachment and automatic error handling for all outgoing API requests.

3. The Instructor Portal

The instructor portal represented the most complex set of UI requirements, translating data-dense workflows into interactive React components.

a. Course and Media Management:

- `AddCourseModal`, `CourseDesign`, and `CourseMDManager` allow instructors to build and manage complete course curriculums with structured module organization and media attachment.
- `CourseDesign` implements a canvas-like interface for arranging course modules in a logical sequence.

b. Dynamic Quiz Authoring:

- `QuizQuestionManager` and `AddQuestionModal` support creation of all Yaksh question types: MCQ, MCC (Multiple Correct Choice), Programming questions with associated test cases, and Arrange-in-order questions.
- **Drag-and-drop question reordering** was implemented within the quiz builder, allowing instructors to arrange questions intuitively without relying on manual index fields.

c. Analytics, Grading, and Live Monitoring:

- `QuizGradingPanel`: Allows instructors to review and override individual student scores.
- `CourseAnalytics.jsx`: Visualizes aggregate performance trends across the student cohort — module completion rates, quiz pass rates, and per-question attempt statistics.
- `QuizMonitorPanel`: Provides near-real-time visibility into which students are currently attempting a quiz and their progress status. The panel consumes `quizMonitorStore` and periodically polls the backend monitoring endpoint to refresh student status data.

4. The Student Portal

The student experience was designed to be distraction-free, maintaining visual consistency with the instructor side to keep the application's theme and structure unified.

a. Active Assessment Interface (`Quiz.jsx` & `Submission.jsx`):

- **Timer synchronization:** The countdown timer state is managed in Zustand to persist across internal question navigation, preventing desynchronization.
- **Autosaving:** Draft answers are saved at configurable intervals to prevent loss of work.
- **Multi-type rendering:** MCQ, code editor, and arrange question types are each rendered in their own appropriate input modes.

- **Submission flow:** A confirmation modal summarizes unanswered questions before final submission to prevent accidental submission.

b. Post-Examination Insights (`Insights.jsx` & `ViewAnswerPaper.jsx`):

- `ViewAnswerPaper` presents a side-by-side diff of the student's submitted code versus the correct reference solution with assertion results — a feature the legacy template-based system did not support, significantly improving post-exam educational value.

c. Course Discussion Forum (`CourseDiscussion.jsx`):

- Provides course-level and lesson-level forum interactions for both students and instructors, allowing threaded posts and comments within the context of a specific course or lesson.

- Forum state is managed by `forumStore.js` and backed by corresponding discussion API endpoints.

5. Theming, Layout, and UI/UX Pipeline

- `ThemeController.jsx` manages application-wide light/dark mode switching, with the user's preference persisted in `authStore`.

- Centralized layout components (`Sidebar.jsx` and `Header.jsx`) enforce DRY (Don't Repeat Yourself) principles, ensuring consistent navigation across all authenticated views.

- All layouts are responsive and adapt for both mobile and desktop viewports, with ARIA attributes and keyboard navigation considered throughout.

6. Testing

- **Vitest + React Testing Library:** Unit and integration tests cover all key flows and components, including UI snapshot tests, form validation, API error handling, permission guards, and child component interaction.

- The test directory structure mirrors the application's component structure for maximum discoverability and maintainability.

5.3 Technical Challenges and Solutions

Challenge 1: Complex Test Case Option Bundling — Data Adapter Pattern

The Problem: A significant architectural mismatch existed between the legacy Yaksh database schema and the React frontend's expected data model for MCQ, MCC, and Arrange question types. The legacy schema stores each answer option as an individual database row, while React components expect a single question object containing a clean nested array of options. MCC questions further required an array of correct-answer indices, and Arrange questions demanded strict sequence preservation — both incompatible with the flat row-per-option database layout.

The Solution — Three-Phase Data Adapter: Rather than migrating the legacy schema (which would have broken existing migrations and grading logic throughout Yaksh), a three-phase adapter was implemented entirely at the API layer, leaving the database completely untouched.

Phase 1 — On Read (GET): The `get_test_cases` method inside `QuestionSerializer` intercepts the flat row list and bundles it into a React-friendly nested array before the response is sent:

Listing 5.1: Read Adapter — Bundling rows into nested array (api/serializers.py)

```
def get_test_cases(self, obj):
    try:
        tc_list = obj.get_test_cases_as_dict()
        if obj.type in ['mcq', 'mcc'] and tc_list:
            mcq_tcs = [tc for tc in tc_list
                       if tc.get('type') == 'mcqtestcase']
            if mcq_tcs:
                first_tc = mcq_tcs[0].copy()
                options_array = []
                correct_data = [] if obj.type == 'mcc' else 0
                for idx, tc in enumerate(mcq_tcs):
                    options_array.append(tc.get('options', ''))
                    if tc.get('correct'):
```

```
                        if obj.type == 'mcc':
                            correct_data.append(idx)
                        else:
                            correct_data = idx
                first_tc['options'] = options_array
                first_tc['correct'] = correct_data
                return [first_tc]
        return tc_list
    except Exception:
        return []
```

Phase 2 — On Write (POST/PUT): When an instructor submits a bundled options array, the view unpacks it back into individual `McqTestCase` or `ArrangeTestCase` database rows:

Listing 5.2: Write Adapter — Unpacking array back into DB rows (api/views.py)

```
if tc_type in ['mcc', 'mcqtestcase']:
    options = tc_data.get('options', [])
    correct_indices = tc_data.get('correct')
    if not isinstance(correct_indices, list):
        correct_indices = [correct_indices] if correct_indices
           else []
    model_class = get_model_class('mcqtestcase')
    for idx, option in enumerate(options):
        if str(option).strip():
            model_class.objects.create(
                question=question,
                options=str(option).strip(),
                correct=(idx in correct_indices),
                type='mcqtestcase'
            )
```

Phase 3 — On Evaluation: When a student submits an Arrange answer (as a comma-separated sequence of 1-based position integers), the view maps these back to the actual `ArrangeTestCase` primary keys that Yaksh's grading engine expects:

Listing 5.3: Evaluate Adapter — Remapping frontend indices to DB IDs (api/views.py)

```
elif current_question.type == 'arrange':
    answer_str = request.data.get('answer', '')
    user_indices = [int(i) for i in answer_str.split(',') if i.
       strip()]
    # Python-side sort to exactly mirror the UI load order
    actual_test_cases = sorted(
        current_question.get_test_cases(), key=lambda x: x.id
    )
    user_answer = []
    for idx in user_indices:
        list_idx = idx - 1  # convert 1-based to 0-based
        if 0 <= list_idx < len(actual_test_cases):
            user_answer.append(actual_test_cases[list_idx].id)
```

This three-phase adapter allowed the React frontend to work with clean, array-driven data structures while the legacy Yaksh database and assessment engine remained completely intact — without a single schema migration.

Challenge 2: Instructor Quiz Testing — Isolated Sandbox Architecture

The Problem: Instructors need to verify that their quiz questions work correctly — exactly as a student would experience them — before publishing. Testing directly against a live course was not viable: instructors are blocked by prerequisite checks, inactive module states, and enrollment validation. Furthermore, any instructor-generated `AnswerPaper` record would be permanently tied to the live course, silently corrupting analytics, average scores, and grading matrices.

The Solution — Transient Sandbox (God Mode): A sandboxed testing architecture was built using FOSSEE's `test_mode(user)` utility, with four interlocking parts:

- **Dynamic Mock Entity Generation:** When the instructor clicks "Test Question", the backend provisions a transient, database-isolated Trial Course, Trial Module, Trial Learning Unit, and Trial Question Paper — all flagged `is_trial=True`.

- **Validation Bypass:** Because the instructor operates inside a Trial Course, all standard student-facing validation gates (enrollment locks, active module checks, prerequisite sequences) are cleanly bypassed:

Listing 5.4: God Mode Bypass Logic (api/views.py)

```
is_trial_mode = course.is_trial and is_moderator(user)
if not is_trial_mode:
    validate_enrollment(user, course)
    check_module_prerequisites(user, module)
```

- **Frontend Execution Parity:** `TestQuestion.jsx` feeds the trial IDs into the same Zustand actions (`quiz_QuestionStore.js`) used by real students — guaranteeing identical runtime behaviour between instructor test runs and actual student sessions.

- **Analytics Isolation:** Standard analytics queries filter out trial courses via `is_trial=False`, so grading summaries and participation dashboards remain unpolluted by instructor QA runs.

5.4 Summary of Contribution

- Designed and implemented the modular Zustand state management architecture with domain-scoped stores for auth, questions, quiz monitoring, regrading, and forum state.

- Built the complete **Instructor Portal**: course and media management, dynamic quiz authoring with drag-and-drop question reordering, analytics dashboard, grading panel, and live quiz monitor.

- Built the complete **Student Portal**: course browsing, distraction-free timer-synchronized quiz interface with autosaving, post-exam `ViewAnswerPaper` diff view, and course discussion forum.

- Implemented `ThemeController` for application-wide light/dark mode with persistent user preference.

- Designed and implemented the **three-phase Data Adapter Pattern** resolving the MCQ/MCC/Arrange schema mismatch between the legacy backend and the React frontend — without any database migration.
- Built the **Isolated Sandbox Architecture (God Mode)** enabling instructors to test questions in full student-parity execution without contaminating live course data or analytics.
- Wrote Vitest + React Testing Library tests covering unit, snapshot, and integration scenarios.
- Delivered the full production build across 147 commits touching 150+ files with approximately 47,000 lines added.

Chapter A

Development Workflow and Timeline

The internship followed an iterative, commit-driven development workflow, with changes made progressively to feature branches that were subsequently reviewed and merged into the main repository. The Git commit history serves as a complete and traceable audit trail of all contributions made across the duration of the project.

A.1 Phase-wise Development Summary

A.2 Repository and Contribution Links

All development work carried out during this internship is publicly traceable through the project's Git history. The key references are as follows:

- **Pull Request:** `https://github.com/FOSSEE/online_test/pull/871`
- **Commit History:** `https://github.com/Mohitranag18/online_test/commits?author=Mohitranag18`
- **Common Repository:** `https://github.com/Mohitranag18/online_test`

The pull request encompasses **147 commits**, touching **150+ files** with approximately **47,000 lines of additions** across the deployment, backend API, frontend SPA, testing, and documentation components of the project.

Phase	Period	Key Activities
Design & Onboarding	Oct – Dec 2025	Study of the existing Yaksh platform and legacy Django template interface; user journey mapping for Instructor and Student roles; wireframing in Figma; high-fidelity prototype creation; design token definition; stakeholder review and design approval; local environment setup with WSL, Miniconda, Redis, and Celery.
Architecture & Backend API	Jan 2026	Headless architecture design; creation of the `api` Django application; serializer development with N+1 query prevention; 250+ RESTful endpoint implementation (student/teacher/common); JWT authentication; question bank API; code execution bridging; analytics and grading endpoints; backend test coverage.
Frontend SPA Development	Feb – Mar 2026	Modular Zustand store architecture; instructor portal (course management, dynamic quiz authoring with drag-and-drop, analytics, grading panel, live quiz monitor); student portal (course browsing, timer-synchronized quiz interface, autosaving, post-exam `ViewAnswerPaper` diff view); course discussion forum; data adapter pattern for MCQ/MCC/Arrange schema mismatch.
QA & Final Refinements	Late Mar – Apr 2026	God Mode / User Mode isolated sandbox architecture for instructor quiz testing; UI fix passes (padding, mobile responsiveness, API crash prevention); Vitest + React Testing Library test suite; final production build delivered in April 2026.

Table A.1: Phase-wise Development Summary

Bibliography

[1] F. Project, "Fossee website," accessed: 2024-12-05. [Online]. Available: https://fossee.in/

[2] ——, "FOSSEE News - January 2018, vol 1 issue 3," accessed: 2024-12-05. [Online]. Available: https://static.fossee.in/fossee/newsletters/Newsletter-Jan2018.pdf

[3] Ministry of Education, Government of India, "National mission on education through ict (nmeict)," 2026, accessed: 2026-05-21. [Online]. Available: https://www.nmeict.ac.in

[4] FOSSEE, IIT Bombay, "Free/libre and open source software for education (fossee)," 2026, accessed: 2026-05-21. [Online]. Available: https://fossee.in

[5] ——, "Yaksh online assessment platform," 2026, accessed: 2026-05-21. [Online]. Available: https://yaksh.fossee.in

[6] FOSSEE, "Yaksh source code repository," 2026, gitHub Repository. [Online]. Available: https://github.com/FOSSEE/online_test

[7] Meta Open Source, "React documentation," 2026, accessed: 2026-05-21. [Online]. Available: https://react.dev

[8] Vite Team, "Vite documentation," 2026, accessed: 2026-05-21. [Online]. Available: https://vitejs.dev

[9] Tailwind Labs, "Tailwind css documentation," 2026, accessed: 2026-05-21. [Online]. Available: https://tailwindcss.com

[10] Django Software Foundation, "Django documentation," 2026, accessed: 2026-05-21. [Online]. Available: https://docs.djangoproject.com

[11] Django REST Framework, “Django rest framework documentation,” 2026, accessed: 2026-05-21. [Online]. Available: https://www.django-rest-framework.org

[12] Zustand, “Zustand documentation,” 2026, accessed: 2026-05-21. [Online]. Available: https://zustand-demo.pmnd.rs

[13] Axios, “Axios documentation,” 2026, accessed: 2026-05-21. [Online]. Available: https://axios-http.com

[14] Figma Inc., “Figma design tool,” 2026, accessed: 2026-05-21. [Online]. Available: https://www.figma.com

[15] Celery Project, “Celery distributed task queue documentation,” 2026, accessed: 2026-05-21. [Online]. Available: https://docs.celeryq.dev

[16] Redis, “Redis documentation,” 2026, accessed: 2026-05-21. [Online]. Available: https://redis.io/docs